SEARCH TERM IDENTIFICATION FOR CONCEPT LOCATION LEVERAGING WORD RELATIONS

by

Shiwen Yang

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2024

Table of Contents

List of Tables

List of Figures

Abstract

According to existing surveys, software errors and failures lead to billion-dollar losses every year. To reduce such losses, software engineers spend about 50% of their time on correcting software errors and bugs. They often use various traditional techniques (e.g., regex, code search) and software artifacts (e.g., bug reports) to detect the location of a software bug, which is time-consuming and technically challenging. Several existing approaches attempt to find appropriate search queries from a bug report (a.k.a., change request), execute them against a search engine, and then detect the location of a software bug (a.k.a., software concept). STRICT, one of the existing works, develops two text graphs leveraging the co-occurrence and syntactic relationships of words and attempts to find the search terms from a change request. However, STRICT overlooks the semantic relationship among the words, which can inform their importance. In this honours thesis, we extend the STRICT and construct a third text graph leveraging the semantic relationship among words. Besides two existing algorithms – TextRank and POSRank, we also replicate three existing algorithms for keyword extraction. We call our solution for keyword selection – STRICT++ – that automatically suggests relevant search terms for a software change request based on four graph-based keyword selection algorithms: POSRank, SimRank, Biased TextRank, and PositionRank. Experiments using 946 change requests from 22 subject systems demonstrate the superiority of STRICT++. It achieves performance improvements, with MAR at 20.78%, MRR at 0.2147, and Top-10 Accuracy at 37.60%, outperforming the STRICT. It provides better first correct ranks than baseline queries for 44%–55% of the change requests which is promising. This honours thesis indicates the significance of capturing syntactic relationships between words for effective keyword extraction. In addition, word similarity, task-specific biases, and position information also play a small supporting role in the combination of graph-based algorithms.

Acknowledgements

I hereby express my deepest gratitude and appreciation to the following individuals and groups who have played a significant role in my academic journey at Dalhousie University.

First and foremost, I extend special thanks to my supervisor, Dr. Masud Rahman, for his invaluable mentorship and guidance. His helpful suggestions provided a firm direction for my research. His insightful feedback and constructive criticism were crucial in shaping my project.

I also wish to express my sincere gratitude to my reader, Dr. Srini Sampalli, for his time and interest in my work. His participation has been invaluable.

To my parents, I owe an immense debt of gratitude for their unwavering support, encouragement, and sacrifices throughout my academic journey. Their love and guidance have been my pillars of strength.

I am also immensely thankful to the members of the RAISE Lab—Sigma Jahan, Ohiduzzaman Shuvo, Parvez Mahbub, Asif Samir, Riasat Mahbub, Usmi Mukherjee, Shihui Gao, Callum MacNeil, and Mehil Shah—for their invaluable help and support. Their expertise and camaraderie have enriched my research experience.

Furthermore, I appreciate Dalhousie University and the Department of Computer Science for fostering an inspiring academic environment and providing the necessary resources for my research.

Lastly, I extend my gratitude to my friends for their guidance and support. Their kindness and encouragement have been invaluable. I am grateful to everyone who has been part of this journey with me, and I look forward to continuing to learn and grow with your support.

Chapter 1

Introduction

1.1 Motivation and Research Problem

During software development and maintenance, software engineers need to deal with many changes or upgrades to their software features. In addition to the changes in software features, they also tackle many software bugs and failures. According to existing surveys, software developers spend about 50% of their programming time dealing with bugs and failures [5]. Locating the bugs in software is an essential step in software debugging, which could be complicated and time-consuming [17] [22] [35]. Similarly, the identification of the exact source location relevant to a change request is equally challenging, which is often called concept location in the literature. Software engineers often attempt to pick appropriate keywords from a change request as queries and execute them with a search engine to find the target code. However, according to previous research findings [9] [16], the success rate of developers selecting appropriate search terms for change tasks is very low, with an average of less than 15%. Thus, supporting the software engineers with appropriate search keywords from change requests - remains an open problem worth studying and solving.

STRICT, a novel search term identification technique to support concept location proposed by Rahman and Roy [25]. STRICT adopts Graph-based Term Weighting where it represents the unique words as vertices and their relationships as edges in a graph. In regular texts (e.g., change requests), words enjoy three major types of relationships, such as statistical (e.g., co-occurrence), syntactic (e.g., grammatical modification), or semantic (i.e., conceptual relevance) [3]. In STRICT, two text graphs were constructed based on term co-occurrence, and POS dependence/syntactic relationships among the words of a change request. They analyze the topological properties from the graphs and extract the search terms by using two graph-based term weighting algorithms, TextRank and POSRank.

Analysis of semantic/conceptual relevance between words has been found useful in search term selection [24]. Rahman and Roy propose QUICKAR that uses the co-occurrence to build a word adjacency database from Stack Overflow. Then it estimates semantic relevance between two words based on their adjacency list from the database to suggest semantically relevant queries for concept location. That is their technique attempts to add highly similar words to change requests. However, adding highly similar keywords might not add any value if the initial query is of poor quality.

Nowadays, graph-based term weighting has emerged as a solution for search term selection. However, existing techniques often consider only one or two inter-word relationships during graph construction, limiting their effectiveness [25] [15] [8]. For instance, while POSRank may perform well in change requests with strong grammar word connections, it may struggle in requests with more code structure and a lack of syntactic relationships. To address this limitation, a combination graph approach can be employed to mitigate the negative effects. In our research, we explore and evaluate the multi-graph-based term weighting method leveraging three types of word relationships for search term selection. We designed three experiments to answer three research questions, aiming to explore the effectiveness of this approach.

1.2 Contributions

In this thesis, we investigate combinations of five graph-based algorithms and their weight distribution for search term selection. We proposed a novel algorithm–STRICT++ that combines four graph-based algorithms (POSRank, SimRank, Biased TextRank, and PositionRank), leveraging three types of inter-word relationships. We conduct experiments on 946 change requests from 22 systems and observe performance improvements over STRICT and baseline queries. The weight distribution of each algorithm underscores the importance of syntactic word relationships, such as POS dependence, in search term selection, as well as the complementary nature of the three inter-word relationships.

1.3 Outline of the Thesis

We conducted three experiments and answered three research questions to explore the impact of five graph-based algorithms leveraging three types of word relationships in keyword selection. This section outlines the different chapters of the thesis.

- Chapter 2 discusses several background concepts (e.g., embedding, cosine similarity, PageRank algorithm) that are required to follow the rest of the thesis.
- Chapter 3 discusses the methodology employed in the study, including the construction of the dataset, text preprocessing, calculation of term weights, search term ranking and selection, parameter tuning, and experimental design.
- Chapter 4 discusses the empirical findings of three experiments (single graph-based algorithm, combination of algorithms, and weight distribution of each algorithm) and answers three research questions.
- Chapter 5 discusses the threats to the validity of our research.
- Chapter 6 discusses the related work.
- Chapter 7 concludes the thesis with some directions for future works.

Chapter 2

Background

This chapter provides the required terminologies and concepts to follow the remaining of the thesis. We introduce key technologies such as word/sentence embeddings, cosine similarity, and degree of interest.

2.1 Embedding

Word embeddings are a key technology for converting words into numerical vector representations. They aim to capture the semantic information of words and encode them into high-dimensional continuous vectors that are easy to process by machine learning models. Word2Vec [21] and GloVe [12] are two pre-trained models that have been used in keyword extraction applications [14] [33].

Sentence embeddings extend the concept of word embeddings and represent an entire sentence or phrase as a single vector, retaining its overall meaning and context. In this work, we use "all-mpnet-base-v2" [30], a pre-trained BERT model from the HuggingFace Model Hub that transforms text into a 768-dimensional dense vector space.

2.2 Cosine Similarity

Cosine similarity is a commonly used technique to determine textual similarity in information retrieval. By computing the cosine similarity between word embeddings, one can directly estimate the semantic relatedness of the corresponding words. The higher cosine similarity values indicate strong semantic overlap. The resulting value ranges from zero (i.e., completely dissimilar) to one (i.e., completely similar). The formula to calculate the cosine similarity between two vectorsis [10] is as follows:

$$Cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|} \tag{2.1}$$

where,

- $x \cdot y$ is the product (dot) of the vectors 'x' and 'y'.
- $\|x\|$ and $\|y\|$ are the magnitudes of the vectors 'x' and 'y'.
- $\|x\|\|y\|$ is a multiplication of two magnitudes of the vectors 'x' and 'y'.

We use cosine similarity to calculate the similarity between words during the graph construction of SimRank and Biased TextRank (section 3.3.3).

2.3 Degree of Interest

When assessing multiple term weighting algorithms, each algorithm offers a unique perspective on term importance. Therefore, normalization is essential to mitigate bias when performing mathematical operations among these graph-based algorithm scores. The normalization formula is as follows:

$$DOI_i = 1 - \frac{n_i}{N} \tag{2.2}$$

where n_i represents its position in the stack trace and N represents the total number of references. This normalization process scales the scores to a range between 0 and 1, allowing for easier comparison. This technique commonly referred to as the 'degree of interest', is widely used in relevant literature [6] [27].

2.4 Summary

The background chapter elucidates the core concepts and techniques employed in the study, including embedding for capturing semantic information (section 2.1), cosine similarity for measuring semantic relatedness (section 2.2), and Degree of Interest (section 2.3). It offers a solid foundation to comprehend the subsequent methodological developments and empirical evaluations.

Chapter 3

Methodology

This chapter details the methodology employed in the study, encompassing the construction of the dataset, text preprocessing, the implementation of five graph-based algorithms, search term selection strategy, parameter tuning, and evaluation metrics.

3.1 Construction of the dataset

We reuse the existing dataset and artifacts of STRICT Replication Package [19], hosted at GitHub. The dataset contains 946 change requests from 22 subject systems, see Table 3.1. They are collected from two popular bug-tracking systems–BugZilla and JIRA. Each change request includes an issue ID, title, and description. Table 3.2 shows an example change request from eclipse.jdt.ui system.

In the experiments, we used not only actual change requests but also solutions applied by developers in practice as ground truth. Each change request is assigned a solution set (i.e., goldset). It is the collection of a changeset (i.e., a list of changed files) for each commit operation from the commit history from the selected projects [25].

Table 3.1: Subject Systems and Associated Query Counts

System	**#Queries**	**System**	**#Queries**
ecf	68	eclipse.jdt.core	29
eclipse.jdt.debug	81	eclipse.jdt.ui	240
eclipse.pde.ui	179	tomcat70	33
adempiere-3.1.0	12	apache-nutch-1.8	9
apache-nutch-2.1	17	atunes-1.10.0	16
bookkeeper-4.1.0	21	commons-math-3-3.0	16
derby-10.9.1.0	24	eclipse-2.0	17
jedit-4.2	10	lang	42
mahout-0.4	16	mahout-0.8	15
math	60	openjpa-2.0.1	16
tika-1.3	18	time	7
Total number of Queries = 946			

Table 3.2: An Example Change Request (Issue #: 303705, ECLIPSE.JDT.UI)

Field	**Content**
Title	[search] Custom search results not shown hierarchically in the java search results view
Description	Consider an instance of org.eclipse.search.ui.text.Match with an element that is neither an IResource nor an IJavaElement. It might be an element in a class diagram, for example. When such an element is reported, it will be shown as a plain, flat element in the otherwise hierarchical java search results view. This is because the LevelTreeContentProvider and its superclasses only check for IJavaElement and IResource.

3.2 Text Preprocessing

Each change request within the dataset is comprised of two primary items: Title and Description. To mitigate the influence of inconsequential or redundant words, we apply a series of standard text preprocessing steps to each change request before their use. We discuss these steps in detail as follows.

3.2.1 Syntactic Segmentation

Syntactic Segmentation is the process of breaking paragraphs into sentences. We employed special punctuations and the newline character (“`\n`”) as delimiters to demarcate sentence boundaries. The sentence segmentation is implemented using the regular expression below [18].

```
(?<=[.?!:;])\s+(?=[a–zA–Z0–9])
```

3.2.2 Term Decomposition

A dotted term (e.g. `org.eclipse.ui.part`) often contains multiple technical artifacts. We split them by dots (i.e. `org`, `eclipse`, `ui`, and `part`). For a CamelCase term (e.g. `createPartControl`), we split into simpler terms, (i.e., `create`, `Part`, and `Control`). We decompose camel case tokens using a regular expression as shown below [18]. This regex helps us find the position of each smaller term, and then we add the space between them for splitting.

```
([a–z])([A–Z]+)
```

Their decomposition into separate components facilitates an individual assessment of the importance of each artifact [7]. Both the original CamelCase form and its decomposed terms are preserved for subsequent analysis, as they may hold distinct informational value in different contexts [25].

3.2.3 Removal of Nonessential terms and Tokens

Certain words, such as *the*, *is*, *and*, *of*, etc., while necessary for constructing grammatically sound sentences, contribute minimal semantic content and recur frequently within texts. These are categorized as **stop words** and are discarded from each change request to attenuate noise. We use a pre-defined list to find the stop words in our research, which is from GitHub [19]. Additionally, we discard **small tokens**, defined as words with fewer than three characters, as they are deemed less informative [25].

We retain the original forms of terms rather than **stemming**, which can more accurately reflect the nuanced semantics and contextual usage prevalent in the specialized language of software change requests [16] [25]. Text preprocessing ensures that subsequent algorithms are performed on a refined and contextually enriched textual representation, thereby enhancing the accuracy of the algorithm's analysis of change requests.

3.3 Calculation of Term Weights

We leverage three types of relationships among words (e.g., statistical, syntactic, semantic) to construct graphs and apply five different algorithms to them to select appropriate keywords from a change request. In the following sections, we discuss them in detail.

3.3.1 Statistical Relationship

Graph Construction

Using Word Co-occurrence: After text preprocessing, we get a set of sentences from each change request. We use these sentences to construct a text graph only based on word co-occurrence. In this graph, vertices represent words, and edges represent co-occurrence relationships between them. We set the window to 2 as recommended by Mihalcea and Tarau [20]. So in each sentence, each group of adjacent words will be connected. The co-occurrence relationship is bidirectional, so the edges on this graph are bidirectional also, see Figure 3.1 (a). Finally, we set the total number of the co-occurrences for each pair of words as their edge weight, which will be used in the PositionRank calculation later.

We consider the sentence-“Custom search results not shown hierarchically in the java search results view“ in Table 3.2 as an example [25]. Then its preprocessed version will be “Custom search hierarchically java search view.” As you can see, there are several phrases such as “custom search” and “search view” semantically depending on each other. Term co-occurrence can capture these dependencies in a statistical sense [25] [23]. For a sliding winder of 2, we can get the following relationships: Custom $\leftrightarrow$ search, search $\leftrightarrow$ hierarchically, hierarchically $\leftrightarrow$ java, java $\leftrightarrow$ search, and search $\leftrightarrow$ view. Each unique word will be the vertice, and connect an edge between each pair in the graph (Figure 3.1 (a)).

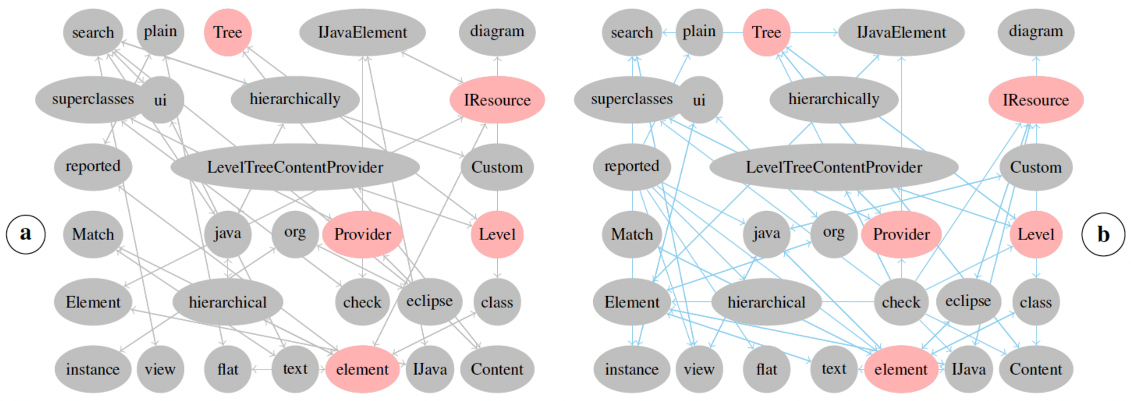

Figure 3.1: Text Graphs of change request in Table 3.2 – (a) using word co-occurrences, and (b) using syntactic dependencies [25]

After constructing a text graph based on word co-occurrence, we use TexRank or PositionRank for term weighting. Their calculation algorithms are followed in the next sections.

TextRank (TR) Calculation

TextRank is one of graph-based techniques for search term identification used by STRICT [25]. It has been adapted from the PageRank algorithm to iteratively compute the score (i.e., importance) for each word. In each iteration, a word's score is updated as a weighted sum of the scores of the words that point to it (recommends). It should be noted that the sum is then divided by the number of outgoing links from the word. The process continues until the scores converge below a specified threshold or reach a maximum number of iterations (default is 1000). Below is the TextRank Algorithm formula that we used:

$$TR(v_i) = (1 - d) + d * \sum_{v_j \in V(v_i)} \frac{TR(v_j)}{|V(v_j)|} \tag{3.1}$$

Here, $V(v_j)$ and $|V(v_j)|$ denote the node list connected to v_i and its total number. d represents the damping factor, which signifies the likelihood of a user selecting a web page in the context of web surfing. Conversely, $1 - d$ is the probability of jumping off that page. The dumping factor has a scalar value between 0 and 1. We use $d = 0.85$ for our TextRank calculation [25].

PositionRank (PTR) Calculation

PositionRank is proposed by Florescu and Caragea [8]. It is a position-biased TextRank, and it has a similar graph structure as TextRank. However, when we connect two candidate tokens, we add the total number of co-occurrences with the document (i.e., change request) as the weight of the edge. The calculation formula is shown below:

$$PTR(v_i) = \widetilde{p_i} * (1 - d) + d * \sum_{v_j \in In(v_i)} \frac{w_{ji}}{\sum_{v_k \in Out(v_j)}} PTR(v_j) \tag{3.2}$$

Here, $In(v_i)$ represents the node list pointing to v_i, while $Out(v_j)$ denotes the node list emanating from v_j. d represents the damping factor (0.85 [8]). w_{ji} is the weight of edge between nodes v_i and v_j. p_i represents the position-biased value of a node, which is the sum of the reciprocals of all positions where the word appears in the text. Therefore, both the position and frequency of the node will affect the value of the p_i value.

3.3.2 Syntactic Relationship

Graph Construction

Using POS Dependence: We construct a text graph based on grammatical dependencies among the words of a sentence. In this graph, vertices again represent words, but edges now denote grammatical modification relationships between them (e.g., subject-verb, verb-object). Jespersen's Rank Theory [13] considers the words in a sentence into three ranks-primary (i.e., nouns), secondary (i.e., verbs, adjectives), and tertiary (i.e., adverbs). The word with a higher rank can define words at the same or lower ranks [13]. So, nouns can modify only nouns, a verb can modify nouns, verbs, or adjectives but not an adverb. We use Stanford POS tagger [32] to annotate words in each sentence and then connect the directed edge based on their syntactic ranks. The direction of the edges reflects the syntactic dependencies of the words.

We consider the sentence–"element reported plain flat element hierarchical java search view" as an example [25]. This sentence can be organized into two ranks–primary ("search", "view", "java", "element"), and secondary ("plain", "flat", "hierarchical", and "reported"). We derive the following relationships based on their grammatical modifications–search $\leftrightarrow$ view, view $\leftrightarrow$ java, java $\leftrightarrow$ element, reported $\rightarrow$ search, reported $\rightarrow$ view, reported $\rightarrow$ java, reported $\rightarrow$ element, reported $\rightarrow$ plain, reported $\rightarrow$ flat, reported $\rightarrow$ hierarchical. and, then encode them into connecting edges in the text graph (i.e., Figure 3.1 (b)) [25].

POSRank (PR) Calculation

POSRank is another graph-based technique for search term identification technique employed by STRICT [25]. It is also an adaptation of PageRank algorithm [4] for natural language texts. Similar to the rank formula of TextRank, POSRank determines the score of nodes based on the connection links. We consider that an incoming link represents a recommendation (or vote) from another term and vice versa. See its formula below:

$$POSR(v_i) = (1 - d) + d * \sum_{v_j \in In(v_i)} \frac{POSR(v_j)}{|Out(v_j)|} \tag{3.3}$$

Here, $In(v_i)$ represents the node list pointing to v_i, while $Out(v_j)$ denotes the node list emanating from v_j. d represents the damping factor (0.85 [25]).

3.3.3 Semantic Relationship

Graph Construction

Using Conceptual Relevance: The construction of the graph relies on the estimation of similarity between every pair of words within each sentence. To do that, we assign each word within the change request to a vector representation first. In our experiment, we use “all-mpnet-base-v2” [30], a pre-trained BERT model from the HuggingFace Model Hub to do word/sentence embedding. Then we employ cosine similarity to quantify the resemblance between these vectors (section 2.2). When the similarity measure value surpasses a predetermined threshold (default set at 0.5), a bidirectional connection is established between the corresponding words in the graph, as depicted in Figure 3.2. In addition, we set their similarity measure value as the weight of the connecting edge, which will be used in the Biased TextRank later. However, because of the potential abundance of words within each change request, the computational complexity for assessing all word pairs could escalate exponentially, consuming substantial processing times. To mitigate this computational burden, we confine our similarity calculations to word pairs within individual sentences, treating each sentence as a textual unit.

We use SimRank and Biased TextRank for term weighting on this text graph. Their calculation algorithms are followed in the next sections.

SimRank (SR) Calculation

In SimRank, we consider whether a term is recommended by another term only based on whether they are connected. Therefore, we use the same term weighting algorithm as TextRank. The calculation formula of the SimRank is as follows:

$$SR(v_i) = (1 - d) + d * \sum_{v_j \in V(v_i)} \frac{SR(v_j)}{|V(v_j)|} \tag{3.4}$$

Here, $V(v_j)$ and $|V(v_j)|$ denote the node list connected to v_i and its total number. d represents the damping factor, and set $d = 0.85$ for our SimRank calculation [25].

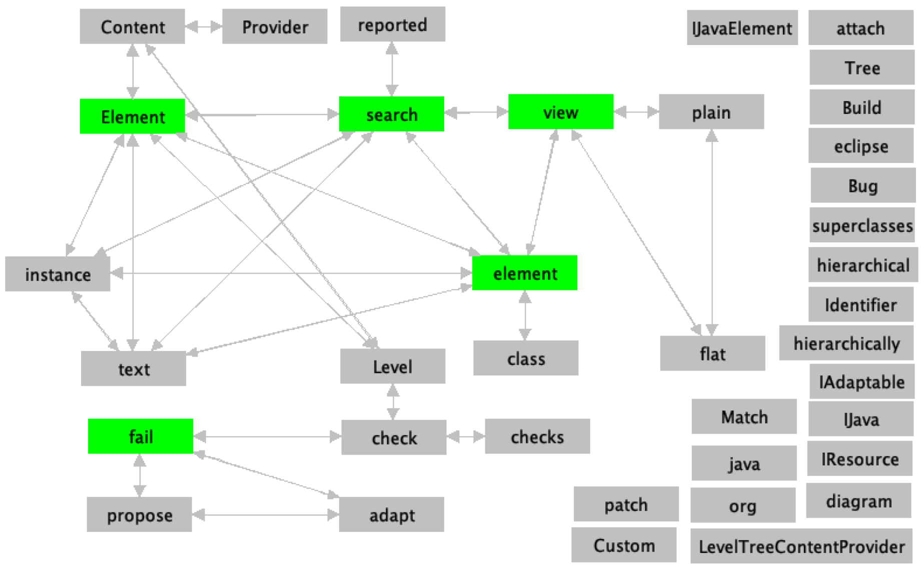

Figure 3.2: SimRank Graph of change request in Table 3.2 using Cosine Similarity (SIMILARITY_THRESHOLD = 0.5)

Biased TextRank (BTR) Calculation

Kazemi et al. [15] introduced Biased TextRank, a graph-based content extraction method that assigns random restart probabilities to nodes in a graph. It gives more weight to the nodes that are closer to the focus text based on the similarity of the graph nodes to the focus of the task. Kazemi et al. [15] applied the Biased TextRank performance to focused summarization and explanation extraction. In the case of keyword extraction, we refine the fundamental unit of analysis from sentences to individual words. Since the title of a change request often serves as a concise encapsulation of its essence, we designate it as the focus (or bias) of the text. On the other hand, candidate search tokens are captured from the description of the change request.

The edge connections are bidirectional with weights in the Biased TextRank graph.

Therefore, Biased TextRank uses the following formula to iteratively update the Biased TextRank score of a node:

$$BTR(v_i) = BiasWeight_i * (1-d) + d * \sum_{v_j \in In(v_i)} \frac{w_{ji}}{\sum_{v_k \in Out(v_j)}} BTR(v_j) \tag{3.5}$$

Here, $In(v_i)$ represents the node list pointing to v_i, while $Out(v_j)$ denotes the node list emanating from v_j. d represents the damping factor (0.85 [15]). w_{ji} is the weight of edge between nodes v_i and v_j. $BiasWeight_i$ represents the relevance of graph node v_i to the focus of the text, which is the similarity measure value between the candidate token and the title in our experiment.

3.4 Search Term Ranking and Selection

In this section, we present and explain our designed algorithm for Search Term Identification, which harnesses Information Retrieval (IR) methods to extract and rank relevant search terms from a given Change Request (CR). The pseudocode of the algorithm, as shown in Algorithm 1, outlines the sequential steps involved in this process.

The algorithm begins by defining a set of ScoreKeyList (Line 2), comprising TR, PR, SR, BTR, and PTR, which correspond to different graph-based scoring methods discussed above (Section 3.3). An empty list L is initialized to hold the identified search terms (Line 3).

The algorithm first extracts essential information from the change request (CR) (Line 4-6). The task title T, and description D are separately collected, followed by their combination and preprocessing, resulting in a unified TD representation. This preprocessing step has been described in section 3.2.

Next, From lines 7 to 11, five distinct text graphs (G_{tr}, G_{pr}, G_{sr}, G_{btr}, G_{ptr}) are constructed from the preprocessed texts, each tailored to capture specific statistical, syntactic, and semantic characteristics of the input text. The detail for each graph is in section 3.3.

Then, we calculate the rank score for all candidate terms in five graphs. For each graph, when running its term weighting algorithm (Line 14), we initialize each of the terms with a default value of 0.25 and run an iterative version of the algorithm [25]. However, the initial term score will not affect its final score [23]. We

Algorithm 1: Search Term Identification using IR Methods

```
1  Input: Change Request $CR$
2  ScoreKeyList ← {TR, PR, SR, BTR, PTR}
3  L ← {} ;                                   // list of search terms
   // collecting task details from the change request
4  T ← collectTitle(CR)
5  D ← collectDescription(CR)
6  TD ← preprocess(combine(T,D))
   // developing text graphs from the task details
7  $G_{tr}$ ← developTRGraph(TD)
8  $G_{pr}$ ← developPRGraph(TD)
9  $G_{sr}$ ← developSRGraph(TD)
10 $G_{btr}$ ← developTSRGraph(EMBED(T), preprocess(D))
11 $G_{ptr}$ ← developTPRGraph(TD)
   // calculating Candidate Token Score
12 TokenScoreMap ← {} ;          // hashmap of candidate token score map
13 for $G \in [G_{tr}, G_{pr}, G_{sr}, G_{btr}, G_{ptr}]$ do
14     CTS ← calculateRankScore(G, scoreKey)
15     $CTS_{norm}$ ← normalize(sortByValue(CTS)
16     TokenScoreMap ← combine(TokenScoreMap, $CTS_{norm}$, scoreKey)
   // determining term importance
17 CST ← getUniqueTerms(TD)
18 for $CandidateSearchTerm\ CST_i \in CST$ do
19     S[$CST_i$] ← calculateTokenRankScore($CST_i$)
   // ranking and then returning Top-K search terms
20 SST ← sortByFinalTermWeight(S)
21 L ← getTopKSearchTerm(SST)
   // add additional title terms
22 foreach $t$ in preprocess(T) do
23     if $t$ is not in $L$ then
24         L.add($t$);
25 return L ;
```

obtain a collection of candidate words accompanied by their respective rank scores in each graph. To mitigate the risk of undue influence exerted by any individual score and ensure a more equitable assessment, we subject normalize the node scores from each graph via the prescribed formula shown in Algorithm 2 (Line 15). Consequently, the resulting normalized scores are invariably confined within the interval [0, 1]. The TokenScoreMap in line 16 is the final candidate term score map to store the rank scores of candidate tokens across the various graph representations.

Algorithm 2: Candidate Score Normalization Algorithm

```
1 Input: R: candidates search terms sorted by scores
2 for CandidateSearchTerm $t \in R.keys$ do
3     R[t] $\leftarrow 1 - \frac{position(t)}{size(R)}$
4 return R ;
```

In line 19, we derive the definitive rank score attributed to each candidate token by aggregating its individual rank scores across all graphs, as outlined in Algorithm 3. To account for the varying impact of individual graph models in shaping the overall term significance, we introduce adjustable parameters $\alpha, \beta, \gamma, \delta$, and ϵ for five different graphs. By default, these weights are set to unity, signifying equal consideration of all graph models (default value is 1). Conversely, should a parameter be assigned a value of zero, it effectively discounts the corresponding graph from the term selection process. The search terms are then ranked based on their final term weights. We select the top K search terms (default value is 10) based on their weights.

Rahman and Roy [25] assigned an additional weight to words appearing within the document title. Title keywords often encapsulate the essence of a change request and thus are suitable candidates for search terms. So, we undertake a final examination to ascertain whether any tokens extracted from the preprocessed title are absent from the generated suggested query (Lines 21-23). If missed, we add the title terms to the ranked search terms to finalize our search query from a change request (Line 24).

Algorithm 3: Calculate Token Rank Score

```
1 Input: candidates search term $CST$
2 $S_{TR} \leftarrow$ TokenScoreMap[CST].textRankScore
3 $S_{PR} \leftarrow$ TokenScoreMap[CST].posRankScore
4 $S_{SR} \leftarrow$ TokenScoreMap[CST].simRankScore
5 $S_{BTR} \leftarrow$ TokenScoreMap[CST].bTextRankScore
6 $S_{PTR} \leftarrow$ TokenScoreMap[$CST_i$].positionRankScore
  // calculating final term-weight
7 S $\leftarrow \alpha S_{TR} + \beta S_{PR} + \gamma S_{SR} + \delta S_{BTR} + \epsilon S_{PTR}$
8 return S ;
```

3.5 Parameter Tuning

The performance of our proposed algorithm is influenced by the values assigned to the tunable parameters α, β, γ, δ, and ϵ. They weigh the contributions of different term weights calculated by different algorithms. To determine optimal settings for these parameters, we implemented an automated parameter tuning process as outlined in Algorithm 4 [16]. We aim to optimize the parameters of a model by iteratively adjusting their values within specified bounds. In particular, we optimize the MRR of our search queries to determine the optimal weights of α, β, γ, δ, and ϵ.

Our parameter tuning process starts with the initial parameter vector, reflecting a priori assumptions about the relative importance of the respective graph algorithms. We check the performance of five term-selection algorithms in Tables 4.1, and start our optimization with the initial values of 2, 3, 2, 2, 1 for α, β, γ, δ, and ϵ respectively. In addition, we also check the performance of distinct combinations of graphs in Table 4.2. The best combinations in each group will also be tested as the initial values.

In each iteration of the optimization process, we consider each parameter individually. By slightly incrementing or decrementing one parameter by a step size (learning rate, λ), we obtain two new sets of parameters (newParamsUp and newParamsDn). The updated values remain within the specified minimum (w_{min}, default is 0) and maximum (w_{max}, default is 100) bounds (Lines 10-11).

Algorithm 4: Tune Parameters

```
1  Input: learning rate $\lambda$, maximum weight $w_{max}$, minimum weight $w_{min}$,
     maximum number of iterations $MAXITER$
2  Output: Optimized parameters
   // Step 1: Initialization
3  Let $Parameters \leftarrow [\alpha = 2, \beta = 3, \gamma = 2, \delta = 2, \epsilon = 1]$
4  Let $fitVal_{best} \leftarrow$ fitness($Parameters$)
   // Step 2: Parameter Refinement
5  Let $itercount \leftarrow 0$
6  while $itercount < MAXITER$ do
7      for $index\ i < Parameters.length$ do
8          Let $newParamsUp \leftarrow Parameters$.clone();
9          Let $newParamsDn \leftarrow Parameters$.clone();
10         $newParamsUp$[i] $\leftarrow$ Math.min($Parameters[i] + \lambda$, $w_{max}$);
11         $newParamsDn$[i] $\leftarrow$ Math.max($Parameters[i]$ - $\lambda$, $w_{min}$);
12         Let $fitVal_{up} \leftarrow$ fitness($newParamsUp$)
13         Let $fitVal_{dn} \leftarrow$ fitness($newParamsDn$)
14         if $fitVal_{up} > fitVal_{best}$ or $fitVal_{dn} > fitVal_{best}$ then
15             Update $fitVal_{best}$ and $Parameters$ ;
16         else
17             Break ;
18     itercount ++
   // Step 3: Output
19 return $\alpha$, $\beta$, $\gamma$, $\delta$, $\epsilon$ ;
```

The default value of the learning rate, λ, is 0.1. To increase the precision of the parameters, i.e., to increase the number of digits after the decimal point, the learning rate is reduced, such as 0.01 or 0.001. Adjusting the learning rate allows for fine-tuning the parameters to achieve higher precision in parameter values, which can be crucial for optimizing the performance of our suggested queries.

The fitness function calculating MRR is evaluated for both the incremented and decremented parameter settings, and the best fitness value is retained. If no improvement in fitness is observed for any parameter or has arrived at the maximum iterations ($MAXITER$) the process stops and returns the current parameter vector as the optimized set. (Lines 12-17)

When training the parameters, we randomly divide the dataset into two distinct subsets: a training set and a testing set. This helps avoid bias caused by overfitting to the specific patterns cases and ensures fairness.

Overall, this algorithm provides a systematic and automated approach to fine-tune the parameters of our technique, potentially improving its performance on a given task.

3.6 Experimental Design

We execute the suggested queries from our technique – STRICT++ algorithm using a Vector Space Model (VSM) based search engine – Apache Lucene [25]. Then we evaluate our queries by comparing their results against the ground truth source files that were changed in response to change requests. In particular, we use four performance metrics and answer three research questions.

3.6.1 Performance Metrics

In our experiment, we employ four performance metrics to rigorously evaluate our suggested search queries. These metrics have been regularly used by relevant research [25] [8], which makes them suitable for our work.

Effectiveness (E):

Effectiveness (E) is the rank of the first true positive in search results. A smaller effectiveness indicates that the query is more efficient and the expected result appears closer to the top of the result list.

Mean Reciprocal Rank (MRR):

Mean Reciprocal Rank (MRR) is a metric that computes the mean of reciprocal ranks of the first correct result within the top-K (K=10) search results for all change requests. The higher MRR values indicate superior query performance. We use this metric not only to evaluate our queries but also to tune our parameters (Section 3.5)

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{FirstGoldRank(q)} \tag{3.6}$$

Here, Q and $|Q|$ denote the set of all queries and their total number respectively. FirstGoldRank(q) is the rank at which the first correct result was found using a query, q.

Mean Average Precision (MAP):

This metric is used to quantify a technique's overall capability to accurately and favorably rank relevant items for multiple queries. It is derived by computing the mean of Average Precision scores across all queries. Average Precision itself is the weighted average of Precision at each occurrence of a relevant item within the top-K results as follows:

$$AP = \frac{\sum_{k=1}^{D} P_k}{|RetrievedGoldset(q)|} \tag{3.7}$$

$$MAP = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{3.8}$$

Here, P_k denotes the precision at k^{th} result. RetrievedGoldset(q) refers to all ranks at which correct results were found using a query (q), and D is the number of the rank list. Q is the set of all queries.

Top-K Accuracy:

It measures the proportion of change requests for which the search queries successfully retrieve at least one true positive result within the top-K results. It serves as a concise, query-based indicator of a technique's ability to accurately retrieve relevant results within high-ranking positions.

3.7 Summary

The methodology chapter outlines a structured and rigorous experimental procedure. Firstly, the chapter introduces the structure of an existing dataset and describes the process of collecting ground truth. Next, the change request is segmented into sentences and pre-processed. Subsequently, five text graphs (TextRank, POSRank, SimRank, Biased TextRank, and PositionRank) are constructed, and the corresponding term weight algorithms are applied to calculate the rank score for each candidate term. The chapter also explains the term ranking algorithm and selection method. Finally, four evaluation metrics are introduced in our experiment. They are Effectiveness(E), Mean Reciprocal Rank(MRR), Mean Average Precision(MAP), and Top-K Accuracy.

Chapter 4

Empirical findings

This chapter presents the empirical results and analyses derived from applying the proposed methodology to the dataset of change requests. It focuses on answering three research questions and assessing the performance of individual graph-based algorithms and their combinations in selecting effective search terms for concept location.

4.1 RQ1: How does each graph help us find effective search terms for concept location?

We evaluate 946 search queries from 22 subject systems in our experiment. Table 4.1 presents their retrieval performance in terms of MAP, MRR, and Top-10 Accuracy, for five different graph-based algorithms: TextRank, POSRank, SimRank, Biased TextRank, and PositionRank.

Table 4.1: Retrieval Performance of Search by Each Algorithm

Query	**MAP**	**MRR**	**Top-10 Accuracy**
TextRank	19.24%	0.2041	33.97%
POSRank	**20.23%**	**0.2135**	**35.30%**
SimRank	19.08%	0.1988	34.79%
Biased TextRank	19.04%	0.2009	34.69%
PositionRank	18.28%	0.1951	32.74%

We see that POSRank performs the best among all techniques in terms of all metrics: MAP (20.23%), MRR (0.2135), and Top-10 Accuracy (35.30%). This indicates that POSRank is the most effective in selecting search terms from a change request. That is, the Parts of Speech (POS) Dependence in the sentence plays a key role in analyzing the keyword extraction. On the contrary, PositionRank has the worst performance in terms of MAP (18.28%), MRR (0.1951), and Top-10 Accuracy (32.74%). Thus, the position and frequency of terms in the change request are less important than other word relationships when determining their importance.

TextRank, SimRank, and Biased TextRank are comparable in performance. Although not as good as POSRank, they still have certain advantages in some indicators. For example, TextRank achieves the second highest MRR value at 0.2041 and MAP value at 19.24%. SimRank has the second highest Top-10 Accuracy value at 34.79%.

So, in the search term selection, the order of these suggested queries based on their performance is:

$$\text{POSRank} > \text{TextRank} \gtrapprox \text{SimRank} \gtrapprox \text{Biased TextRank} > \text{PositionRank}$$

When tuning parameters, we use the performance ranking of individual algorithms as a reference for setting weights or adjusting related parameters of these algorithms. POSRank gets the highest weight (3) because of its superior performance, followed by TextRank, SimRank, and Biased TextRank gets the second highest weight (2), while PositionRank weights 1 (section 3.5, RQ3).

4.2 RQ2: How do various combinations of graphs help us find effective search terms for concept location?

In the second experiment, we examine the performance of each combination of graphs in search term selection. We analyze the combinations of varying numbers of graph algorithms (from 2 to 5) and evaluate their queries in terms of MAP, MRR, and Top-10 Accuracy.

When TextRank (TR), PositionRank (PR), SimRank (SR), Biased TextRank (BTR), and POSRank (PTR) are used simultaneously, their combination achieves a MAP of 19.99%, an MRR of 0.2084, but a relatively low Top-10 Accuracy of 33.76%. This is slightly worse than the combination of fewer text graphs, suggesting that in search term selection, more text graphs and inter-word relationships might always not lead to better performance. As shown in our experiment, too many factors can introduce noise and reduce the overall performance. In the following sections, we investigate the combination of less than five graph-based algorithms.

4.2.1 Four-graph combinations

As shown in Table 4.2, the performance of various four-graph combinations varies significantly. Among them, the combination of TR, PR, SR, and BTR attains the

Table 4.2: Retrieval Performance of Graph Combination

TR	**PR**	**SR**	**BTR**	**PTR**	**MAP**	**MRR**	**Top-10 Accuracy**
Five Graphs							
✓	✓	✓	✓	✓	19.99%	**0.2084**	33.76%
Four Graphs							
✓	✓	✓	✓		19.87%	0.2062	**34.71%**
✓	✓	✓		✓	19.13%	0.2012	33.88%
✓	✓		✓	✓	18.93%	0.1997	33.11%
✓		✓	✓	✓	18.97%	0.1961	32.99%
	✓	✓	✓	✓	**20.06%**	**0.2116**	33.92%
Three Graphs							
✓	✓	✓			20.47%	0.2118	**35.87%**
✓	✓		✓		19.26%	0.2015	35.52%
✓	✓			✓	19.08%	0.2014	33.80%
✓		✓	✓		19.38%	0.2008	34.29%
✓		✓		✓	20.02%	0.2083	34.77%
✓			✓	✓	18.50%	0.1923	33.26%
	✓	✓	✓		**20.79%**	**0.216**	35.03%
	✓	✓		✓	19.31%	0.2021	33.34%
	✓		✓	✓	19.95%	0.2094	33.23%
		✓	✓	✓	19.24%	0.1993	32.69%
Two Graphs							
✓	✓				19.32%	0.2017	**36.32%**
✓		✓			**20.75%**	**0.2138**	35.51%
✓			✓		18.63%	0.1936	33.34%
✓				✓	18.77%	0.2022	34.13%
	✓	✓			20.42%	0.2114	35.42%
	✓		✓		20.35%	0.2134	34.79%
	✓			✓	18.58%	0.1994	32.40%
		✓	✓		18.53%	0.1960	33.88%
		✓		✓	19.38%	0.2013	31.80%
			✓	✓	18.18%	0.1920	32.82%

TextRank(TR), POSRank(PR), SimRank(SR), Biased TextRank(BTR), and PositionRank(PTR) highest Top-10 Accuracy of 34.71%, with relatively balanced MAP (19.87%) and MRR (0.2062) scores. On the other hand, the combination excluding TR (i.e., PR, SR, BTR, PTR) has the highest MAP and MRR values, at 20.06% and 0.2116 respectively with a second highest Top-10 Accuracy of 33.92%. The remaining four-graph combinations do not demonstrate notable performances in search term selection.

4.2.2 Three-graph combinations

In three-graph combinations, the exclusive use of TR, PR, and SR stands out with excellent performance across all evaluation metrics. Specifically, it achieves the highest Top-10 Accuracy at 35.87% among three-graph combinations, along with the second highest MAP (20.47%) and MRR (0.2118) scores. Additionally, the combination using only PR, SR, and BTR reaches the optimal MAP value of 20.79% and optimal MRR value of 0.216. Similarly, a combination of PR, SR, and BTR is found to be the best combination when only MAP and MRR are considered. After comparing the performance of three-graph combinations with combinations of two, four, and five graphs, we find that using three text graphs may be an optimal choice for a text-graph combination algorithm.

4.2.3 Two-graph combinations

The performance of two-graph combinations is mixed. The combination using only TR and SR has the second highest Top-10 Accuracy (35.51%), and its MRR (0.2075) and MRR (21.38%) are both the best in this group. The combination of TR and PR achieves the highest Top-10 Accuracy among all combinations at 36.32%, but its MAP and MRR are not very high. Other two-graph combinations do not exhibit clear advantages across all metrics.

4.2.4 Summary

The performance of different graph combinations in search term selection varies significantly, with no ultimate winner. However, a few combinations demonstrate exceptional performance:

- The combination of POSRank, SimRank, and Biased TextRank demonstrates strong performance across all metrics, achieving optimal MAP and MRR values of

20.79% and 0.216, respectively, along with a high Top-10 Accuracy of 35.03%. The MRR value of 0.216 indicates that, on average, the first correct result appears at position 1 / 0.216 $\approx$ 4.63 in the search result list. This means that users need to scroll down to approximately the fourth to fifth results to find the result.

- The combination of TR and SR attains the highest Top-10 Accuracy among all combinations at 36.32%, but its MAP and MRR are both lower than the average values.

These findings indicate that judicious selection and a combination of specific graph algorithms can effectively enhance performance in search term selection performance. In practice, depending on task requirements and resource constraints, the appropriate graph algorithm combination should be chosen based on their evaluation metrics. TR, PR, SR, and BTR demonstrate strong synergies in different combinations. In addition, both POSRank and SimRank are included in the top two optimal combinations. Thus, POSRank and SimRank potentially serve as potent tools for optimizing keyword selection systems.

4.3 RQ3: Can we find optimal weights for various algorithms targeting search term selection?

In this experiment, we optimized term selection from change requests using five graph-based algorithms–TextRank, POSRank, SimRank, Biased TextRank, and PositionRank–and determined the relative importance of their scores through parameter tuning, see Algorithm 4 in section 3.5. We attempt to find a combination of weights for these algorithms, which can outperform the STRICT [25].

Table 4.3 shows several results of weighted graph combinations in parameter tuning. The best weight combination found in our experiment is TR (TextRank) = 0, PR (PositionRank) = 9, SR (SimRank) = 0.4, BTR (Biased TextRank) = 1, and PTR (POSRank) = 1 achieving an MRR of 0.2181. This indicates that, on average, the first correct result appears at position 1 / 0.2181 $\approx$ 4.58 when users need to find the most relevant result. We will refer to this new technique, with this specific weight contribution, as STRICT++.

The POSRank algorithm has the highest weight (9), underscoring the importance of syntactic relationships between words in keyword extraction. Although SimRank

Table 4.3: Performance of Weighted Graph Combination in Parameter Tuning

Initialization Parameters						**Output Weights and Metric**					
$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$	$\boldsymbol{\delta}$	$\boldsymbol{\epsilon}$	**Learning rate ($\boldsymbol{\lambda}$)**	**TR**	**PR**	**SR**	**BTR**	**PTR**	**MRR**
2	3	2	2	1	0.1	1	4	2.1	2	1	0.2101
1	1	1	1	1	0.1	0	2	0.2	1.1	0.9	0.2170
0	1	1	1	1	0.1	**0**	**9**	**0.4**	**1**	**1**	**0.2181**
0	1	1	1	0	0.1	0	11	1	1	0.2	0.2177
1	0	1	0	0	0.1	0	8	0	0	1	0.2170

α, β, γ, δ, and ϵ = Initialization weights of TR, PR, SR, BTR, and PTR in Algorithm 4;
TR, PR, SR, BTR, and PTR = The weights of TextRank, POSRank, SimRank, Biased TextRank, and PositionRank;

Table 4.4: Comparison of retrieval performance between STRICT and STRICT++

Query	**MAP**	**MRR**	**Top-10 Accuracy**
STRICT [25]	19.32%	0.2017	36.32%
STRICT++	**20.78%**	**0.2147**	**37.60%**

is included in STRICT++, its low weight (0.4) suggests a relatively limited impact on the experiment. In contrast, both Biased TextRank and PositionRank carry higher weights (both 1), indicating significant roles for task-specific biases and position biases in search term selection. TextRank, however, was excluded (weight 0). We can find the reason from the graph construction outlined in section 3.3.1. Both PositionRank and TextRank graphs are constructed based on word co-occurrences. However, PositionRank incorporates more biases than TextRank, including the position information of words, effectively integrating TextRank principles. As a result, PositionRank carries a higher weight than TextRank. Similar reasoning applies to the comparison between SimRank and Biased TextRank.

The experimental results are almost consistent with the theoretical expectations. In STRICT++, POSRank, Biased TextRank, and PositionRank have a more significant impact on search term selection compared to the other two graph-based algorithms. These three algorithms leverage statistical, syntactic, and semantic word relationships respectively, highlighting the importance of considering all three types of word relationships in search term selection.

Table 4.4 compares the performance of STRICT++ and STRICT. STRICT++ outperforms STRICT with a MAP of 20.78%, MRR of 0.2147, and Top-10 Accuracy of 37.6%.

4.3.1 Comparison with Baseline Queries

When addressing software change requests, a common practice among developers is to copy text directly from change requests and apply it to source code searches. Hence, such change requests effectively serve as the baseline queries against which the performance of our suggested queries is evaluated and validated [2] [11] [16]. Our experiment encompassed three distinct baseline query types. These were: Title, Description, and a composite Title+Description. Additionally, a truncated Title (10 keywords) is also considered, which reflects our algorithmic constraints to truncate the query length to a maximum of 10 keywords.

We collected the ranks of the first correct result for each software change request using STRICT++ and each baseline query. Subsequently, we calculated the percentages of cases where STRICT++ showed improvement, deterioration, or maintenance of the ranking status quo compared to the baselines. Another comparative evaluation metric is the Mean Rank Difference (MRD), which quantifies the arithmetic mean of the discrepancies in rank values observed between STRICT++ and the baseline queries across the entire change requests. A negative MRD value (e.g., -215) signifies that the first correct result achieved by STRICT++ is retrieved at an upper-rank position than that of the baseline queries, thereby indicating improved search efficiency.

Table 4.5: Comparison of STRICT++'s Effectiveness with Baseline Queries

Query Pairs	**Improved**	**Worsened**	**Preserved**	**MRD**
STRICT++ vs. Title	44.69%	32.91%	22.40%	-85
STRICT++ vs. Title (10 keywords)	54.67%	23.99%	21.34%	-215
STRICT++ vs. Description	44.06%	34.29%	21.66%	-125
STRICT++ vs. (Title+Description)	38.54%	38.64%	22.82%	4

MRD = Mean Rank Difference between STRICT++ and baseline queries

Table 4.5 provides a comprehensive comparison of the efficacy of STRICT++ against the performance of the baseline queries. It reveals that our suggested queries outperform the baseline queries in 44% to 55% of cases when compared with the Title, Title (10 keywords), and Description queries while preserving quality in 21%

to 23% of cases. The big negative MRD values also prove that the STRICT++ can return an earlier rank position than the baseline queries, which further underscores the potential superiority of our proposed approach. However, the performance of STRICT++ and composite Title+Description queries are almost the same. Although 38.54% of change requests are improved with the help of STRICT++, 38.64% of change requests return a later rank position. The value of MRD (4) is small and negligible. It is noteworthy that STRICT++ exhibits a marked enhancement in performance relative to the 10-keyword Title variant. When comparing against the unrestricted Title baseline query, STRICT++ not only boosts an additional 10% of change requests to have an upper first correct rank position but also realizes the average ranking improvement represented by MRD is more than twice as large.

Thus, compared with the baseline queries, our suggested query technique STRICT++ is slightly more effective than baseline queries from the change requests. They provide better ranks than baseline queries for 44%–55% of the requests which is promising.

4.4 Summary

The empirical findings chapter presents a comparative evaluation of five graph-based algorithms (TextRank, POSRank, SimRank, Biased TextRank, and PositionRank) in terms of MAP, MRR, and Top-10 Accuracy. The results show that POSRank performs well not only on its own but also when combined with other algorithms. This highlights the importance of POS dependence in keyword extraction. In addition, when processing change requests with less syntax structure, Biased TextRank, and PositionRank complement POSRank by providing semantic and position information. We propose a novel technique, STRICT++, which combines POSRank, SimRank, Biased TextRank, and PositionRank, leveraging three types of word relations. STRICT++ achieves performance improvements outperforming STRICT.

Chapter 5

Threats to validity

This study has employed rigorous methods to evaluate TextRank, POSRank, SimRank, Biased TextRank, and PositionRank in search term selection for concept location. However, we should acknowledge several potential threats to validity to ensure a comprehensive understanding of the results and their implications.

5.1 Internal Validity

Threats to internal validity relate to experimental errors and biases [16]. The accuracy and efficiency of the algorithms under study may be influenced by the specific re-implementation details. Although we have made every effort to adhere to the published specifications of STRICT (TextRank and POSRank) [25], Biased TextRank [15], and PositionRank [8], variations in coding practices, libraries, or computational resources could introduce subtle differences in performance. To mitigate this threat, we have used well-established and widely adopted libraries for graph-based computations and ensured thorough testing of our implementations to mitigate this threat.

The choice of similarity thresholds can impact the performance of SimRank and Biased TextRank. We have performed an ablation study in RQ2 and found that when set 0.5 as the similarity threshold can get better performance in the combination of different text graphs. However, there may still be alternative configurations that yield better results in specific contexts. During the parameter tuning, different weights for the SimRank and Biased TextRank may cause a new optimal similarity threshold value. Future work could explore more exhaustive parameter sweeps or adaptive tuning strategies to further optimize the algorithms' performance.

5.2 External Validity

Threats to external validity concern the generalizability of a technique [28]. The results obtained are contingent on the characteristics of the datasets used for experimentation. Variations in factors such as change requests, source code structure, and the presence of specific domain terminology may impact the generalizability of our findings. Despite our efforts to enhance diversity by employing multiple subject systems, some systems, like apache-nutch-1.8 and Time, have fewer datasets. The distribution of change requests per subject system is uneven. Additionally, our experimentation was limited to Java-based systems [19], which could introduce bias into the results.

During the re-implementation of Biased TextRank, we employed the title as the bias phrase for queries and performed keyword extraction from descriptions. This approach assumes that the title is more significant and represents common or reasonable practices in the industry. While these assumptions serve as a practical starting point for evaluation, actual developer behavior may deviate from these simplified assumptions. Conducting user studies or analyzing real-world query logs could offer a more accurate assessment of the relative benefits provided by our proposed methods.

It is essential to consider that STRICT++ relies on precise parsing of syntactic relations in change requests. However, many change requests may be hastily drafted by non-technical users, which could lead to requests with unclear or chaotic grammatical structures. In this case, the part-of-speech tagger will struggle to identify the syntactic connections necessary for constructing the graph in POSRank. Consequently, when faced with change requests containing grammatically flawed text, STRICT++'s ability to accurately discern and rank search terms can be compromised. Although STRICT++ has support from the word co-occurrence and conceptual relevance, the high contribution of syntactic relation still causes this threat.

Chapter 6

Related work

The concept of search term identification has been widely studied in software engineering, with various approaches proposed. This chapter delves into the related work that inspired and contrasts with the research presented in this thesis.

6.1 TF-IDF

Determining the relative importance of terms in a text document is often called term weighting [29]. Two term weighting methodologies are commonly used in software engineering. In addition to the graph-based approach of STRICT++, the other method is frequency-based. TF-IDF [31] is a frequency-based term weighting. Unlike graph-based term weighting, TF-IDF considers the importance of a term solely based on its frequency within a document and its rarity across the entire corpus [29]. This means that a term that is frequent within a document but rare in other files across the corpus is considered as the keyword for that document. TF-IDF has been widely adopted by the literature for term weighting [11] [16]. However, TF-IDF does not explicitly consider the syntactic roles of words or their semantic associations, which are integral to STRICT++'s approach. The optimal search keywords might not always be frequent in the change request [29]. So, TF-IDF's simplicity does not inherently account for the complex interdependencies among words that may exist in the specialized domain of software engineering change requests.

6.2 QUICKAR

Rahman and Roy [24] proposed a novel technique QUICKAR. It makes good use of two candidate word sources, Stack Overflow and the source code. Stack Overflow is used to build a word adjacency list database for keyword extraction by performing standard natural language preprocessing and capturing co-occurring words. In the algorithm, QUICKAR uses the cosine similarity measure for the contextual similarity between words for candidate terms from the project and checks the co-occurrence frequency between words for candidate terms from Stack Overflow to calculate the R score. This technique gave me a lot of inspiration when I tried to extend STRICT [25]. Both QUICKAR and the extended technique STRICT++ in the project use semantical relevance. However, the difference is that QUICKAR correlates the candidate words with the words in the database built based on Stack Overflow, to find words similar to the candidate words and add them to the suggested search terms. STRICT++ does not use other resources but only focuses on the change request itself. Perform a similarity search in the content of the change request to explore the possible relationship between semantic relevance and suggested search words.

6.3 BLIZZARD

Rahman and Roy [26] proposed a novel technique BLIZZARD. This technique is primarily intended to address the challenge that bug reports may contain different types of structured information. Whether lacking rich structured information or having too much-structured information, can lead to poor search results. Therefore, BLIZZARD divides the bug reports into three categories according to different structural elements before keyword extraction: stack traces, program elements, and natural language. Then correspond to three different graphs: Trace Graph, Text Graph, and Source Term Graph. Among them, text graph development for bug reports with program elements is exactly the TextRank and POSRank graphs that STRICT [25] used word Co-occurrences and POS dependencies. In addition, in the development of source-term graphs for bug reports using only natural language. Pseudo-relevance feedback related to similarity is mentioned. The main purpose of using pseudo-relevance feedback is to supplement bug reports with appropriate keywords, just like the similarity database used in QUICKAR [24], which plays a complementary role. However, in our project, STRICT++ detects the similarity of word pairs to give them different weights instead of looking for more similar terms.

6.4 ChatGPT

The advent of large language models like OpenAI's ChatGPT [34] presents a contrasting approach to search term identification and concept location in software engineering. Unlike our information retrieval principle-based algorithm, ChatGPT is a large-scale neural network model that learns from vast amounts of text data to generate coherent and sometimes creative outputs. Although ChatGPT is primarily used for natural language understanding and generation tasks, it is a potential competitor in search term selection.

Although ChatGPT possesses powerful capabilities in natural language understanding and generation, it presents different trade-offs in cost-effectiveness and determinism compared to STRICT++. Lage language models typically require substantial computational costs for both training and inference due to their complex neural network architectures. However, information retrieval techniques often rely on more straightforward indexing and query processing mechanisms. In addition, ChatGPT can generate multiple possible outcomes based on input due to their probabilistic nature, which may suggest novel and contextually relevant search keywords in some cases. However, it may also result in inaccurate generation or content irrelevant to actual change requirements, known as the "hallucinated" phenomenon [34]. In contrast, STRICT++ is cheaper and more deterministic, consistently producing valid search terms without the risk of generating irrelevant content.

The integration of two methodologies would leverage their respective strengths: Information retrieval-based algorithms provide precise and reliable keyword suggestions based on established inter-word relationships and term weighting; ChatGPT contributes additional contextually relevant keywords or phrases based on its strong contextual understanding and creative output capabilities.

6.5 Summary

Term weighting is one of the methods to determining the keywords in a text document. Besides the graph-based term weighting in STRICT++, the frequency-based term weighting method–TF-IDF has traditionally served as a foundational tool for extracting relevant keywords from textual artifacts (e.g. bug reports and change requests). The quality of change requests significantly impacts the performance of information retrieval (IR)–based search keyword selection. Therefore, some literature (e.g. QUICKAR, BLIZZARD) explores extending the suggested queries by analyzing semantic relatives to supplement context with appropriate keywords. ChatGPT is also a potential competitor with its strong contextual analysis capability in search term selection.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Research indicates that developers face difficulties in selecting search terms based on a change request. An existing technique–STRICT–helps keyword selection and suggests search tokens using two term-weighting techniques–TextRank and POSRank. However, this technique misses using the semantic relationship among words from three types of inter-word relationships to implement text graphs. We designed a SimRank graph using the semantic relationship among words. In addition, Biased TextRank and PositionRank are introduced. Experiments with 946 change requests from 22 subject systems on the combination of five text graphs show the importance of all three types of word relationships contributing to search term selection. We proposed a new technique called STRICT++, a weighted multi-graph-based keyword selection tool leveraging three different word relationships. The improved retrieval performance of STRICT++ over STRICT and baselines substantiates the efficacy of the weight distribution on distinct text graphs in search term selection for concept location. Experimental data along with supporting materials are available elsewhere [1].

7.2 Future Work

Future work could focus on optimizing TextRank's contribution or further fine-tuning the weights to enhance performance. Additionally, since we only experimented with Java-based systems, there is an opportunity to generalize STRICT++ to other language subject systems. Furthermore, change requests are primarily written in natural language text, which causes the high contribution of POSRank (syntactic word relationship) in our new technique. We should evaluate STRICT++ on change requests with more code structure and less grammar to explore its performance in all structure types of change requests.

Bibliography

[1] STRICT++: Experimental Data. URL https://github.com/Lareina-Y/STRICT-QR-Module.

[2] Blake Bassett and Nicholas A. Kraft. Structural information based term weighting in text retrieval for feature location. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 133–141, 2013.

[3] Roi Blanco and Christina Lioma. Graph-based term weighting for information retrieval. *Information Retrieval*, 15(1-2):54–92, 2012. doi: 10.1007/s10791-011-9172-x.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[5] CAMBRIDGE, UK (PRWEB). Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year Share Article, January 2013. [Online] https://www.prweb.com/releases/2013/1/prweb10298185.htm – Last accessed: 2013-01-08.

[6] Joel Cordeiro, Bruno Antunes, and Paulo Gomes. Context-based recommendation to support problem solving in software development. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*, pages 85–89. IEEE, 2012.

[7] Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. Can better identifier splitting techniques help feature location? In *2011 IEEE 19th International Conference on Program Comprehension*, pages 11–20, 2011.

[8] Corina Florescu and Cornelia Caragea. PositionRank: An unsupervised approach to keyphrase extraction from scholarly documents. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1115. Association for Computational Linguistics, July 2017.

[9] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987.

[10] GeeksGorGeeks. Cosine Similarity. [Online] https://www.geeksforgeeks.org/cosine-similarity/ – Last accessed: 2023-02-17.

[11] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. Automatic query reformulations for text retrieval in software engineering. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 842–851, 2013.

[12] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation, August 2014. [Online] https://nlp.stanford.edu/projects/glove/ – Last accessed: 2023-02-24.

[13] Otto Jespersen. *The philosophy of grammar*. Routledge, 2013.

[14] Ning Jianfei and Liu Jiangzhen. Using word2vec with textrank to extract keywords. *Data Analysis and Knowledge Discovery*, 32(6):20–27, 2016.

[15] Ashkan Kazemi, Verónica Pérez-Rosas, and Rada Mihalcea. Biased textrank: Unsupervised graph-based content extraction. *arXiv preprint arXiv:2011.01026*, 2020.

[16] Katja Kevic and Thomas Fritz. Automatic search term identification for change tasks. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, page 468–471, 2014.

[17] Tien-Duy B. Le, Richard J. Oentaryo, and David Lo. Information retrieval and spectrum based bug localization: better together. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 579–590. Association for Computing Machinery, 2015.

[18] Masudur Rahman. STRICT-QR-Module, 2021. [Online] https://github.com/masud-technope/STRICT-QR-Module – Last accessed: 2023-01-20.

[19] Masudur Rahman. STRICT-Replication-Package, 2021. [Online] https://github.com/masud-technope/STRICT-Replication-Package – Last accessed: 2023-01-20.

[20] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[22] Chris Parnin and Alessandro Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, page 199–209. Association for Computing Machinery, 2011.

[23] Mohammad Masudur Rahman and C. K. Roy. Textrank based search term identification for software change tasks. In *Proc. SANER*, pages 540–544, 2015.

[24] Mohammad Masudur Rahman and Chanchal K. Roy. Quickar: Automatic query reformulation for concept location using crowdsourced knowledge. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 220–225, 2016.

[25] Mohammad Masudur Rahman and Chanchal K. Roy. Strict: Information retrieval based search term identification for concept location. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 79–90, 2017.

[26] Mohammad Masudur Rahman and Chanchal K. Roy. Improving ir-based bug localization with context-aware query reformulation. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 621–632. Association for Computing Machinery, 2018.

[27] Mohammad Masudur Rahman, Shamima Yeasmin, and Chanchal K Roy. Towards a context-aware ide-based meta search engine for recommendation about programming errors and exceptions. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 194–203. IEEE, 2014.

[28] Mohammad Masudur Rahman, Chanchal K. Roy, and David Lo. Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 349–359, 2016.

[29] Mohammad Masudur Rahman, Foutse Khomh, Shamima Yeasmin, and Chanchal K Roy. The forgotten role of search queries in ir-based bug localization: An empirical study. *Empirical Software Engineering*, 26(6):116, 2021.

[30] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[31] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 60(5):493–502, 2004.

[32] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 human language technology conference of the North American chapter of the association for computational linguistics*, pages 252–259, 2003.

[33] Yujun Wen, Hui Yuan, and Pengzhou Zhang. Research on keyword extraction based on word2vec weighted textrank. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pages 2109–2113. IEEE, 2016.

[34] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.

[35] Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. How practitioners perceive automated bug report management techniques. *IEEE Transactions on Software Engineering*, 46(8):836–862, 2020.